Introduction

This is the specifications document for the NLP Stack. There are two principal components this project. The first is the NLP Stack, which does the model development, document and text processing, data cleaning, and more. The second is the NLP Website, which handles project management, text classifications, and is the user interface.

These two systems take vastly different resources and so this document splits them into sections. There is significant interaction, however, and the database will end up tightly coupled to the NLP Stack. While PostgreSQL is not explicitly needed, a SQL database with stored procedures as outlined in the NLP schema is. Since I provide the NLP Schema, I would just recommend using PostgreSQL.

The Software

I developed the NLP Stack using Java 8 (and 10), Ubuntu 18.04, and Maven. In addition, this platform will require a SQL database. While it does not require PostgreSQL but I did design the NLP Schema based around it. I would recommend PostgreSQL 10.*.

I made the database a self-contained entity. The stored procedures should be sufficient to manage any component interaction. In short, models should likely not exist in any application and should rely on stored procedures for all database interactions.

In addition, PostgreSQL prevents SQL injection attacks by preventing SQL injection. This includes all dynamic SQL and in all cases where SQL injection is incredibly useful (such as using base SQL to create identical tables with different names). The safest way to manage this is to simply use the database for models, migration, and stored procedure interaction.

We created the website using Laravel and PHP. It connects to the database using a standard authenticated user and relies on stored procedures for any and all database access.

The NLP Stack

The NLP Stack does two separate things. The first is that it brings in data and the second is that it produces output. Data sources can vary but I standardize all data regardless of format, split texts into sentences, tag words, spell check, remove punctuation, and put all text back into the database.

Importing data relies heavily on the OpenNLP stack, particularly for spell checking. A collection processing engine reads the data into the UIMA framework. The framework distributes the pieces accordingly.

Given the size of the data, I heavily optimize and index the PostgreSQL tables.

This requires a fairly big server. I would recommend at least 100GB of storage, 4 cores, and 16-32GB of ram per project. Stress testing will solidify the specifications but assume a minimum of 100GB for storage.

The NLP Stack will use all resources at its disposal. For larger data sets, I recommend at least 16 cores with 64GB of RAM or more. For huge data sets, we might think about spark clusters. For data sets on the order of 1GB, I recommend at least 8 cores with 32GB of ram for the initial read. Storage is not important as the database handles that. More RAM is almost universally better in this phase.

I include various output options as part of this process. Some of these options take very little effort. Some take more. I will specifically address model creation.

Creating models is computationally intensive. 200 models with very small training sets takes approximately 5 hours on a four-core machine. I suspect that we would distribute model creation across a variety of machines, once we hit thresholds, or we will simply wait the hours it takes to process. I store all of the model files within the database for versioning and historical reference. I have plans in place to only update models when training data changes. That said, the more cores I have available, the more models I generate at the same time. RAM is not a bottleneck at this stage. Even 2GB or so per core would be useful.

The Website

This is entirely dependent on the number of supported users. Given that we are going to have a small handful, I would expect a very small server. It doesn't require huge RAM requirements, apart from maintaining connections, the database handles the storage, and the CPU simply has to run the web server. That said, as the number of users increases, so will the server and database capabilities.

The website will hit the database on almost all postbacks since each postback will call a stored procedure. This process could get quite slow but further testing will reveal stresses on the database side and where we can and should improve things.